AC Chapter 4

Created	@November 23, 2023 3:53 PM
Class	AC
Reviewed	☐

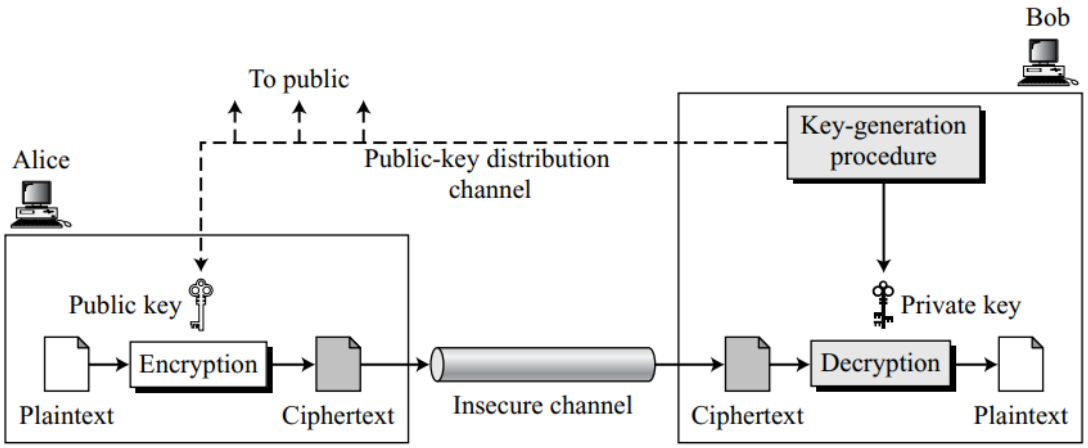
Public Key Cryptography

Asymmetric key cryptography uses two separate keys: one private and one public.

Unlike in symmetric-key cryptography, plaintext and ciphertext are treated as integers
in asymmetric-key cryptography. The message must be encoded as an integer (or a set
of integers) before encryption; the integer (or the set of integers) must be decoded into
the message after decryption.

Encryption and decryption in asymmetric-key cryptography are mathematical functions
applied over the numbers representing the plaintext and ciphertext. The ciphertext can be
thought of as C = f (Kpublic, P); the plaintext can be thought of as P = g(Kprivate, C).

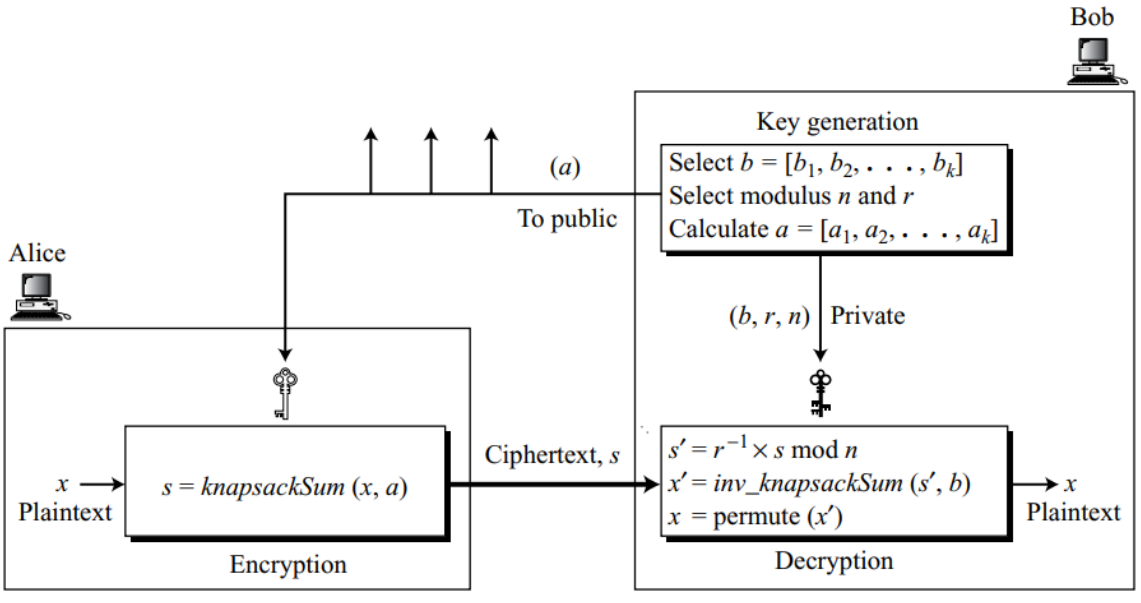
Figure 10.2 *General idea of asymmetric-key cryptosystem*



Knapsack Cryptosystem

The first idea of public-key cryptography came from Merkle and Hellman, in their knapsack cryptosystem.

Figure 10.4 *Secret communication with knapsack cryptosystem*



Key Generation

a. Create a super-increasing k-tuple b = [b1, b2, …, bk]. This is the private key.

b. Choose a modulus n, such that n > b1 + b2 + … + bk

c. Select a random integer r that is relatively prime with n and 1 ≤ r ≤ n −1.
d. Create a k-tuple a = [a1, a2,…, ak] in which ai = r × bi mod n. This is the public key.

Encryption

a. A converts his message to a k-tuple x = [x1, x2,…, xk] in which xi is either 0 or 1. The tuple x is the plaintext.
b. C = xi * ai, C is the cipher text. Cycle the public key multiple times if message is longer than public key. A sends B the ciphertext C.

Decryption

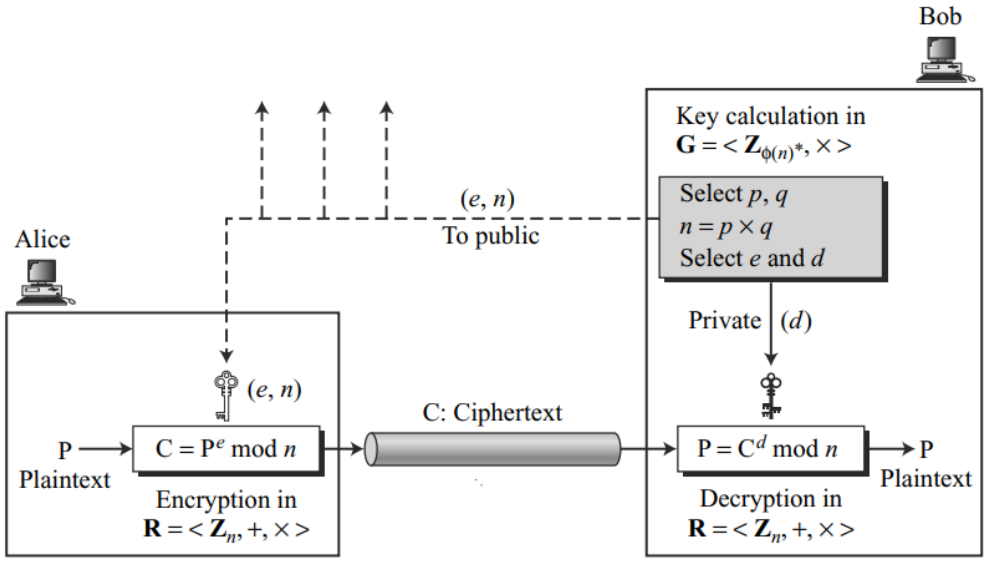
B receives ciphertext C.
a. B calculates s′ = r^−1 × s mod n.
b. Bob solves the knapsack problem to create x based on private key.
The tuple x is the recovered plaintext.

RSA Cryptosystem

The most common public-key algorithm is the RSA cryptosystem, named for its inventors (Rivest, Shamir, and Adleman).

RSA uses two exponents, e and d, where e is public and d is private. Suppose P is the
plaintext and C is the ciphertext. Alice uses C = (P^e) mod n to create ciphertext C from
plaintext P; Bob uses P = (C^d) mod n to retrieve the plaintext sent by Alice. The modulus n,
a very large number.

Figure 10.6 *Encryption, decryption, and key generation in RSA*



RSA Key Generation Algorithm:

1. Select two large primes p and q such that p ≠ q.
2. n ← p × q
3. φ(n) ← (p − 1) × (q − 1) //Euler's Totient Function
4. Select e such that 1 < e < φ(n) and e is coprime to φ(n)
5. d ← (e^−1) mod φ(n) // d is inverse of e modulo φ(n)
6. Public_key ← (e, n) // To be announced publicly
7. Private_key ← d // To be kept secret

In RSA, p and q must be at least 512 bits; n must be at least 1024 bits.

Encryption

C = (P^e)%n

Decryption

P = (C^d)%n

Attacks on RSA

1. Factorization Attack:
   The security of RSA is based on the idea that the modulus is so large that it is infeasible to factor it in a reasonable time. Bob selects p and q and calculates n = p × q.
   Although n is public, p and q are secret. If Eve can factor n and obtain p and q, she

can calculate φ(n) = (p − 1) (q − 1). Eve then can calculate d = e−1 mod φ(n) because
e is public.

2. Chosen-Ciphertext Attack
   A potential attack on RSA is based on the multiplicative property of RSA. Assume that
   Alice creates the ciphertext C = (P^e) mod n and sends C to Bob. Also assume that Bob
   will decrypt an arbitrary ciphertext for Eve, other than C. Eve can calculate P back from C.

3. Attacks on the Encryption Exponent
   To reduce the encryption time, it is tempting to use a small encryption exponent e. The
   common value for e is e = 3 (the second prime). However, there are some potential
   attacks on low encryption exponent.

4. Common Modulus Attack

   The common modulus attack can be launched if a community uses a common modulus, n. For example, people in a community might let a trusted party select p and q, calculate n and φ(n), and create a pair of exponents (ei, di) for each entity. Now assume Alice needs to send a message to Bob. The ciphertext to Bob is C = (P^eB) mod n. Bob uses his private exponent, dB, to decrypt his message, P = (C^dB) mod n. The problem is that Eve can also decrypt the message if she is a member of the community and has been assigned a pair of exponents (eE and dE)

Recommendations

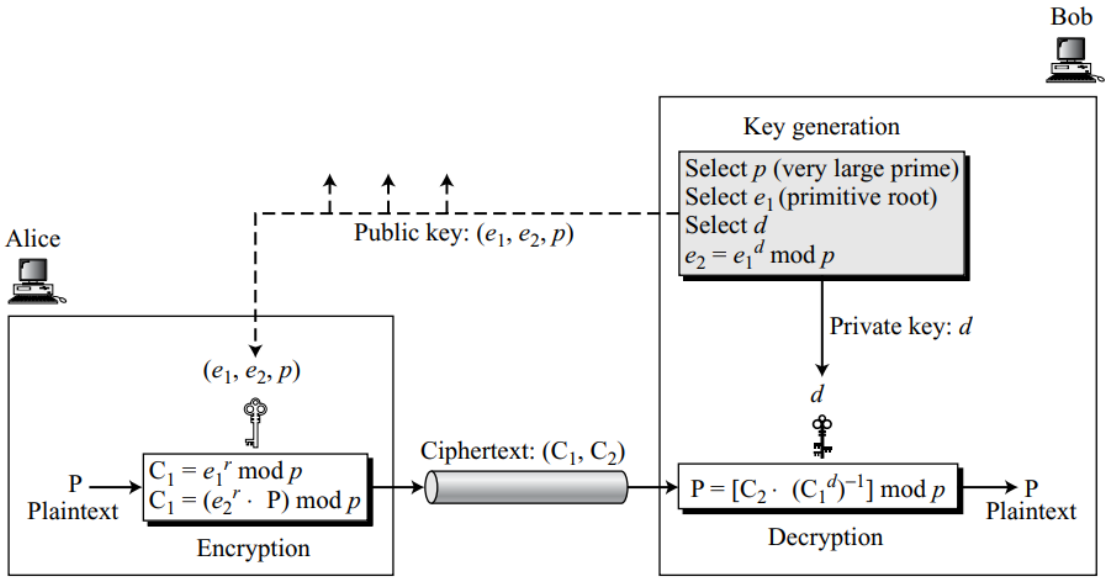
The following recommendations are based on theoretical and experimental results.

1. The number of bits for n should be at least 1024. This means that n should be around 21024, or 309 decimal digits.
2. The two primes p and q must each be at least 512 bits. This means that p and q should be around 2512 or 154 decimal digits.
3. The values of p and q should not be very close to each other.
4. Both p − 1 and q − 1 should have at least one large prime factor.

5. The ratio p/q should not be close to a rational number with a small numerator or denominator.
6. The modulus n must not be shared.
7. The value of e should be 216 + 1 or an integer close to this value.
8. If the private key d is leaked, Bob must immediately change n as well as both e and
   d. It has been proven that knowledge of n and one pair (e, d) can lead to the discovery of other pairs of the same modulus.

El Gamal Cryptosystem

Figure 10.11 *Key generation, encryption, and decryption in ElGamal*



Key Generation

1. Select a large prime p
2. Select d to be a member of the group G = < Zp*, × > such that 1 ≤ d ≤ p − 1
3. Select e1 to be a primitive root in the group G = < Zp*, × >
4. e2 ← (e1^d) mod p
5. Public_key ← (e1, e2, p) // To be announced publicly
6. Private_key ← d // To be kept secret

Encryption

1. Select a random integer r in the group G = < Zp*, × > Means between 1 and p-1.
2. C1 ← (e1^r) mod p
3. C2 ← (P × e2^r) mod p // C1 and C2 are the ciphertexts

Decryption

$$P \leftarrow [C_2 (C_1^{d})^{-1}] \bmod p \qquad \text{// P is the plaintext}$$

Attacks on El Gamal Cryptosystem

Low-Modulus Attacks

If the value of p is not large enough, Eve can use some efficient algorithms (see Chapter 9) to solve the discrete logarithm problem to find d or r. If p is small, Eve can easily find $d = \log_{e1} e_2 \bmod p$ and store it to decrypt any message sent to Bob.This can be done once and used as long as Bob uses the same keys. Eve can also use the value of C_1 to find random number r used by Alice in each transmission $r = \log_{e1} C_1 \bmod p$. Both of these

Known-Plaintext Attack

If Alice uses the same random exponent r, to encrypt two plaintexts P and P′, Eve discovers P′ if she knows P. Assume that $C_2 = P \times (e_2^{r}) \bmod p$ and $C'_2 = P' \times (e_2^{r}) \bmod p$. Eve finds P′ using the following steps:

1. $(e_2^{r}) = C_2 \times P^{-1} \bmod p$
2. $P' = C'_2 \times (e_2^{r})^{-1} \bmod p$

It is recommended that Alice use a fresh value of r to thwart the known-plaintext attacks.

For the ElGamal cryptosystem to be secure, p must be at least 300 digits and r must be
new for each encipherment.

Diffie-Hellman Key Agreement

In the Diffie-Hellman protocol two parties create a symmetric session key.

Before establishing a symmetric key, the two parties need to choose two numbers p and g.

The first number, p, is a large prime number on the order of 300 decimal digits (1024 bits).

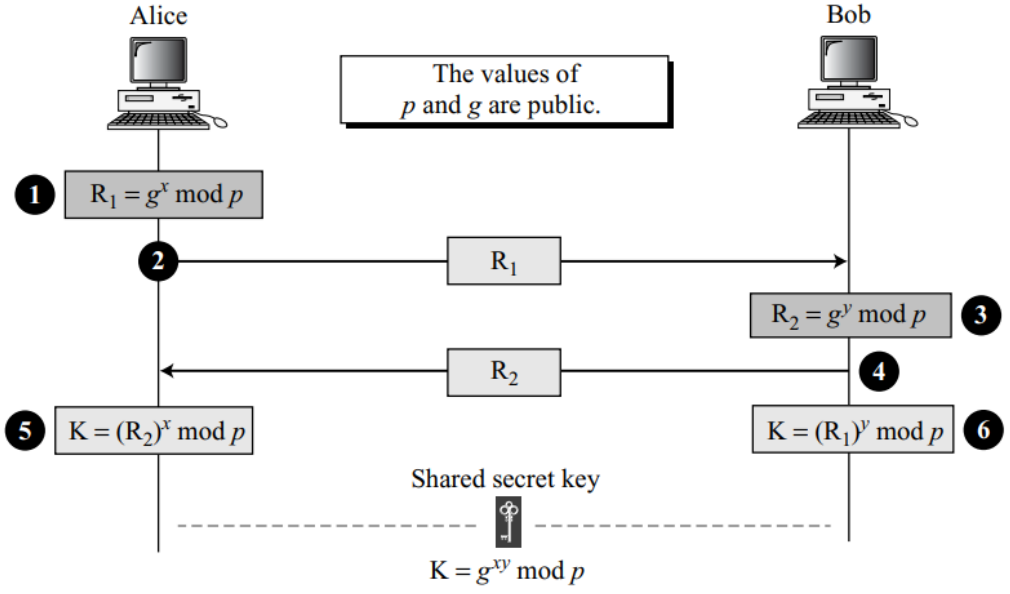
The second number, g, is a generator of order p − 1 in the group <Zp*, ×>.

These two (group and generator) do not need to be confidential.

The steps are as follows:

1. Alice chooses a large random number x such that $0 \le x \le p - 1$ and calculates $R_1 = g^x \bmod p$.
2. Bob chooses another large random number y such that $0 \le y \le p - 1$ and calculates $R_2 = g^y \bmod p$.
3. Alice sends R_1 to Bob. Note that Alice does not send the value of x; she sends only R_1.
4. Bob sends R_2 to Alice. Again, note that Bob does not send the value of y, he sends only R_2.
5. Alice calculates $K = (R_2)^x \bmod p$.
6. Bob also calculates $K = (R_1)^y \bmod p$.

Figure 15.9 *Diffie-Hellman method*



The symmetric (shared) key in the Diffie-Hellman method is K = (g^xy) mod p.

Security of Diffie-Hellman

The Diffie-Hellman key exchange is susceptible to two attacks: the discrete logarithm attack and the man-in-the-middle attack.

1. Discrete Logarithm Attack: The security of the key exchange is based on the difficulty of the discrete logarithm problem. Eve can intercept R1 and R2. If she can

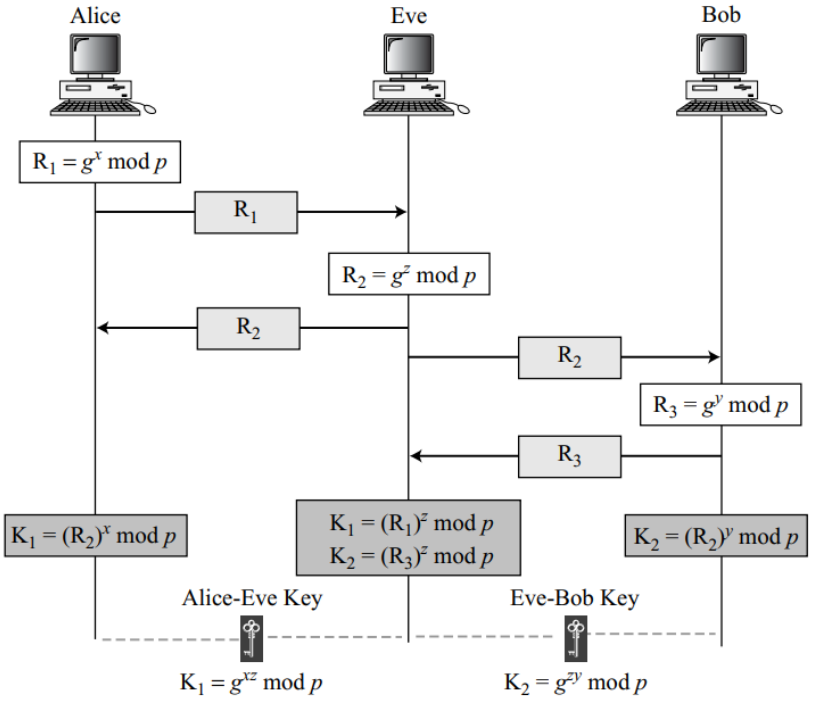
find x from R1 = (g^x) mod p and y from R2 = (g^y) mod p, then she can calculate the symmetric key K = (g^xy) mod p. The secret key is not secret anymore.

2. Man-in-the-Middle Attack:

   The following can happen:

   1. Alice chooses x, calculates R1 = (g^x) mod p, and sends R1 to Bob.
   2. Eve, the intruder, intercepts R1. She chooses z, calculates R2 = (g^z) mod p, and sends R2 to both Alice and Bob.
   3. Bob chooses y, calculates R3 = (g^y) mod p, and sends R3 to Alice. R3 is intercepted
      by Eve and never reaches Alice.
   4. Alice and Eve calculate K1 = (g^xz) mod p, which becomes a shared key between Alice
      and Eve. Alice, however, thinks that it is a key shared between Bob and herself.
   5. Eve and Bob calculate K2 = (g^zy) mod p, which becomes a shared key between Eve
      and Bob. Bob, however, thinks that it is a key shared between Alice and himself.

Figure 15.11 *Man-in-the-middle attack*



To make Diffie-Hellman safe from the discrete logarithm attack, the following are recommended.

1. The prime p must be very large (more than 300 decimal digits).
2. The prime p must be chosen such that p − 1 has at least one large prime factor (more than 60 decimal digits).
3. The generator must be chosen from the group <Zp*, × >.
4. Bob and Alice must destroy x and y after they have calculated the symmetric key.
   The values of x and y must be used only once.